CST8177 – Linux II

bash startup files
Linux/Unix files
stty

Todd Kelley
kelleyt@algonquincollege.com

Topics

- midterms (Feb 27 and April 10)
- bash startup files
- More Linux Files review
- stty

Configuring Bash Behavior

- We customize our shell behavior by
  - setting environment variables, for example,

    ```
    export PATH=/bin:/usr/bin:/sbin
    ```
  - setting aliases, for example

    ```
    alias ll="ls -l"
    ```
  - setting shell options, for example,

    `shopt -s failglob` or `shopt -s dotglob`
  - setting shell options, for example,

    ```
    set -o noclobber
    ```

we make these customizations permanent using bash startup files

Bash Startup Files

- http://teaching.idallen.com/cst8207/13f/notes/210_startup_files.html
- `~/.bash_profile` is sourced by your login shell when you log in
  - the things we set up here are done only once when we log in
  - `export`-ed variables here are inherited by subshells
  - we `source ~/.bashrc` here because login shells do not source it
- `~/.bashrc` is sourced by each non-login subshell, interactive or not
  - here we set up things that are not inherited from the login shell
  - inside this file, at the top, we check whether it's an interactive or non-interactive shell:

```
[ -z "${PS1-}" ] && return
```

  - we set aliases in this file
  - we set options configured with `shopt` and `set` in this file

Startup File Sequence

- When a login shell starts
  1. execute commands from `/etc/profile`, if that file exists
  2. execute commands from the first of these that is readable (in order):
     1. `~/.bash_profile`
     2. `~/.bash_login`
     3. `~/.profile`
- When a login shell exits
  1. read and execute commands from the file `~/.bash_logout`, if it exists

Startup File Sequence (cont'd)

- When an interactive non-login shell starts
  1. execute commands from `~/.bashrc`, if that file exists
- The `--rcfile` <u>`file`</u> option specifies that <u>`file`</u> should be used instead of `~/.bashrc`

System Wide Shell Configuration

- The system administrator can configure the default shell environment for all users
- Configuration in `/etc/profile` applies to all users on the system
- The files in `/etc/skel/` are copied to newly created user accounts (can give new users a default copy of `.bash_profile` and `.bashrc`)

Non-Interactive Shells

- The bash process used to execute a shell script is non-interactive
- `stdin` and `stdout` not connected to a terminal (more details in bash manpage)

.bashrc versus .bash_profile

- .bash_profile is loaded once by a login shell
- .bashrc is loaded by non-login shells
- There are cases where there never is a login shell, for example

```
ssh remote-server.com <some_command>
```

- So the method we'll use in this course:
  - `.bash_profile` does nothing except load .bashrc
  - `.bashrc` keeps track of things that should be done only once

.bashrc

```
[ -z "${PS1-}" ] && return
if [ "${_FIRST_SHELL-}" = "" ] ; then
    export _FIRST_SHELL=$$
    export PATH="$PATH:$HOME/bin"
    export LC_ALL=en_CA.UTF-8
    export LANG=en_CA.UTF-8
    # here we put things that
    # should be done once
fi
# here we put things that need to be
# done for every interactive shell
```

.bash_profile

Contains just one line:

```
source ./.bashrc
```

Unix Files

- Sobel, Chapter 6
- 160_pathnames.html *Unix/Linux Pathnames (absolute, relative, dot, dot dot)*
- 450_file_system.html *Unix/Linux File System – (correct explanation)*
- 460_links_and_inodes.html *Hard links and Unix file system nodes (inodes)*
- 460_symbolic_links.html *Symbolic Links – Soft Links – Symlinks*
- 500_permissions.html *Unix Modes and Permissions*
- 510_umask.html *Umask and Permissions*

Information given by long listing: ls -l

10 characters

- file type as the first letter
- access modes (remaining letters)

Link count

- number of links to this file or directory

User-owner Login Name

- user who owns the file/directory
- based on owner UID

User-owner Group Name

- group who owns the file/directory
- based on owner GID

File Size

- size (in bytes or K) of the file/directory

Date/Time Modified

- date and time when last created / modified / saved

File Name

- actual file/directory name

File Types

- Linux recognizes and identifies several file types, which is coded into the first letter of the first field of information about the file:

-	(**dash**)a regular file
b	block device special file
c	character device special file
d	a directory
l	a symbolic (soft) link
p	a named pipe or FIFO
s	socket special filename

File Access Privileges

- In Linux, 3 types of access permissions or privileges can be associated with a file:
  - **read** (r) grants rights to read a file
  - **write** (**w**) grants rights to write to, or change, a file
  - **execute** (**x**) grants rights to execute the file (to run the file as a command)
- All 3 permissions can then be applied to each of 3 types of users:
  - **User:** owner of the file
  - **Group:** group to which user must belong to gain associated rights
  - **Others:** not **User** and not member of **Group** (sometimes called “World” or “Everybody”)

Octal representation of permissions

r w x	***Octal Value***	***Meaning***
0 0 0	0	No permission
0 0 1	1	Execute-only permission
0 1 0	2	Write-only permission
0 1 1	3	Write and execute permissions
1 0 0	4	Read-only permission
1 0 1	5	Read and execute permissions
1 1 0	6	Read and write permissions
1 1 1	7	Read, write and execute permissions

Directory Access Privileges

- The same three types of access permissions or privileges are associated with a directory, but with some differences:
  - **read** (**r**) rights to read the directory
  - **write** (**w**) rights to create or remove in the directory
  - **execute/search** (**x**) rights to access the directory

  meaning, cd into the directory, or access inodes it contains, or "pass through"

All three permissions can then be applied to each of three types of users as before.

  - **User** owner/creator of the file
  - **Group** group to which user must belong
  - **Others** everyone else (Rest-of-world)

Linux File Permissions

- Three special access bits. These can be combined as needed.
- SUID - Set User ID bit
  - When this bit is set on a file, the effective User ID of a process resulting from executing the file is that of the owner of the file, rather than the user that executed the file
  - For example, check the long listing of /usr/bin/passwd – the SUID bit makes this program run as root even when invoked by a regular user – allowing regular users to change their own password

chmod 4xxx file-list

chmod u+s file-list

Linux File Permissions

- SGID - Set Group ID bit
  - Similar to SUID, except an executable file with this bit set will run with effective Group ID of the owner of the file instead of the user who executed the file.

chmod 2xxx file-list

chmod g+s file-list

Linux File Permissions

- sticky bit (restricted deletion flag)
    - The sticky bit on a directory prevents unprivileged users from removing or renaming a file in the directory unless they are the owner of the file or the directory
    - for example, **/tmp** is a world-writeable directory where all users need to create files, but only the owner of a file should be able to delete it.
    - without the sticky bit, hostile users could remove all files in /tmp; whereas with the sticky bit, they can remove only their own files.

chmod 1xxx dir-list

chmod +t dir-list

Linux File Permissions

- The permissions a user will have is determined in this way:
  - If the user is the owner of the file or directory, then the **user** rights are used.
  - If the user is not the owner but is a member of the group owning the file or directory, then the **group** rights are used.
  - If the user is neither the owner nor a part of the group owning the file, then the **other** rights are used.
- NOTE: It is possible to give the “world” more permissions that the owner of the file. For example, the unusual permissions `-r--rw-rw-` would prevent only the owner from changing the file – all others could change it!

umask

- The permissions assigned to newly created files or directories are determined by the **umask** value of your shell.
- Commands:
  - **umask** - display current umask
  - **umask xyz** - sets new umask to an octal value **xyz**
- permissions on a newly created file or directory are calculated as follows:
  - start with a “default” of 777 for a directory or 666 for a file
  - for any 1 in the binary representation of the umask, change the corresponding bit to 0 in the binary representation of the default
- umask is a reverse mask: the binary representation tells you what bits in the 777 or 666 default will be 0 in the permissions of the newly created file or directory

umask examples (Files)

- if umask is 022
  - binary umask representation: 000010010 = 022
  - default file permissions 666: 110110110
  - permissions on new file: 110100100 = 644
- if umask is 002
  - binary umask representation: 000000010 = 002
  - default file permissions 666: 110110110
  - permissions on new file: 110110100 = 664
- if umask is 003
  - binary umask representation: 000000011 = 003
  - default file permissions 666: 110110110
  - permissions on new file: 110110100 = 664

umask examples (Files, cont’d)

- notice that for files, a umask of 003 ends up doing the same thing as a umask of 002
- Why?

umask examples (Directories)

- if umask is 022
  - binary umask representation: 000010010 = 022
  - default dir permissions 777: 111111111
  - permissions on new dir : 111101101 = 755
- if umask is 002
  - binary umask representation: 000000010 = 002
  - default dir permissions 777: 111111111
  - permissions on new dir : 111111101 = 775
- if umask is 003
  - binary umask representation: 000000011 = 003
  - default dir permissions 777: 111111111
  - permissions on new dir : 111111100 = 774

umask examples (Dirs, cont'd)

- notice that for directories, a umask of 003 gives different results than a umask of 002
- Why?

Linux File Permissions

- It is important for the Linux file system manager to govern permissions and other file attributes for each file and directory, including
  - ownership of files and directories
  - access rights on files and directories
  - The 3 timestamps seen in **stat (man stat)**
- The information is maintained within the file system information (**inodes**) on the hard disk
- This information affects every file system action.

Linux Basic Admin Tools

- **chown owner[:group] files**
  - Change ownership of files and directories (available for **root** only)

Examples:

chown guest:guest file1 dir2

- change ownership of **file1** and **dir2** to user **guest** and group **guest**

chown guest dir2

- change ownership of **dir2** to user **guest** but leave the group the same

chown :guest file1

- change ownership of **file1** to group **guest** but leave the user the same (can also use **chgrp**)

Linux Basic Admin Tools

- **chmod permissions files**
  - Explicitly change file access permissions

Examples:

chmod +x file1

- changes **file1** to have executable rights for user/group/other, subject to umask

chmod u+r,g-w,o-rw file2

- changes **file2** to add read rights for user, remove write rights for group and remove both read and write rights for others

chmod 550 dir2

- changes **dir2** to have only read and execute rights for user and group but no rights for other

New Commands: stty

- recall the effect of these control characters:
  - `^Z` suspend the current foreground process
  - `^C` terminate the current foreground process
  - `^D` end of file character
  - `^U` kill character to erase the command line
- these are actually properties of the terminal
- they can be set with the stty command
- `stty -a` : print out the current tty settings
- `stty susp ^X` :(that's a caret ^, shift-6 on my keyboard, followed by capital X) means set the susp character to CTRL-X instead of CTRL-Z

stty (cont'd)

- if you accidentally dump the contents of a binary file to your screen, and all the control characters reconfigure your terminal on you, you can reset it to sane values with

```
      stty sane
```